Empty seat finding service using object detection: 'Empty Seats?'

Doyeop Kim[1], Jaeyoon Park[2], Dayeon Woo[3], and Jimin Choi[4]

[1] Sungkyunkwan University, Department of Advanced Materials Science & Engineering kimdobby@g.skku.edu
[2] Sungkyunkwan University, School of Electronic and Electrical Engineering jiwoo0110@g.skku.edu
[3] Sungkyunkwan University, Department of Data Science wuvele22@g.skku.edu
[4] Sungkyunkwan University, Department of Culture and Technology jiminijr@g.ssku.edu

Abstract. 'Empty Seats?' combines the YOLOv5 computer vision model with data processing algorithms, such as OpenCV perspective transform, to provide students with real-time information on the availability of seats in campus spaces. The system captures video or real-time camera feeds, divides them into frames, and uses YOLOv5 to detect objects and determine seat occupancy. The processed information is then forwarded to a server, stored in a database, and delivered through a React-based front-end interface. This streamlined process not only makes it easier for students to find study spaces but also optimizes campus resource utilization, significantly reducing the time and effort needed to locate available seats.

Keywords: Computer Vision, Perspective Transform, Real-Time Data, Node Server, Web Service

1 Introduction

The 'Empty Seats?' application innovatively addresses the challenge of locating available seating in campus spaces at Sungkyunkwan University's Natural Science Campus. With its expansive campus, students often expend significant time and energy in search of a suitable study spot. Except outside areas like the SKKU Central/Samsung Library, which have a seat reservation system, this is particularly true. To alleviate this issue, 'Empty Seats?' integrates the YOLOv5 computer vision model with a user-friendly interface, significantly enhancing the efficiency of finding study spaces.

The primary concern addressed by 'Empty Seats?' is the absence of real-time seating availability information. This lack leads to several challenges: students often visit multiple locations in vain, wasting time and energy; space administrators lack insights into space utilization; and the existing platforms providing facility information are fragmented and space-specific, complicating the search process.

'Empty Seats?' is developed across three key domains: AI, frontend, and backend. The AI component employs YOLOv5 for detecting humans and objects in various spaces, delivering timely updates about seat occupancy. This data is refined using the OpenCV library for 2D transformation, allowing for accurate matching of seat positions. The frontend, built with React, offers a streamlined UI/UX, enabling users to easily navigate through a list of campus spaces and view real-time seat occupancy. This interface includes an introduction page, a comprehensive list of spaces, and detailed pages showing seat occupancy and layout diagrams. The backend architecture encompasses both a database and a server, responsible for storing and transmitting space-related information to the frontend.

In essence, the 'Empty Seats?' app is a holistic solution, starting from capturing video feeds, processing these through YOLOv5 and OpenCV, and then transmitting this processed data for storage and display. This system is poised to significantly enhance the campus experience by providing real-time, accurate seating information, thereby minimizing the time and effort students invest in finding study spaces.

2 Design for the Proposed System

2.1 Design & Implementation Summary

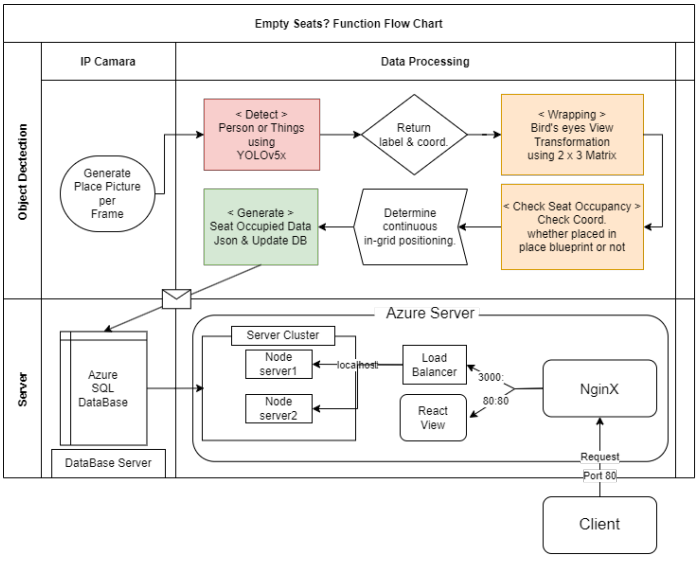

Fig. 1. Service Logic Flow Chart

The overall implementation architecture, depicted in the figure 1 , bifurcates into two primary components: AI modules focusing on object detection using YOLOv5 and OpenCV, and a Docker container encapsulating the application built with React, integrated with an Azure SQL database and server.

The AI module initiates with an IP camera that captures and transmits real-time data. Subsequently, each frame from the video feed is processed through the YOLOv5x model. Opting for the largest model variant was necessary to enhance accuracy, which was found lacking in smaller models. These frames are then transformed using the OpenCV library to a top-down perspective, employing a 2D perspective transform. This approach simplifies the matching of seat positions with detected objects. A customized Python algorithm then processes this information, deducing seat occupancy on a predefined seat layout. The resulting data are formatted into JSON and transmitted to the server.

The application segment is divided into frontend and backend parts. The frontend, developed using React and MUI materials, offers a straightforward and intuitive UI/UX. It encompasses four main screens: an introductory page, a list of available spaces, a detailed page showing seat occupancy and layout, and an information page about each space with descriptive hashtags. Efficient page routing and connectivity are managed using React Router DOM, while MUI materials facilitate the inclusion of components like navigation bars, enhancing user interaction.

In the backend, the Azure SQL database and server act as conduits between the AI modules and the frontend. The database is designed for minimal resource utilization and rapid communication, storing essential information such as space names, addresses, descriptions, hashtags, and crucially, seat occupancy status. The server's role involves hosting APIs to facilitate data fetching between the app and database. Docker is employed to compartmentalize these instances into individual containers, enabling the simultaneous operation of the application and server.

2.2 Skill & Technique

YOLOv5 & OpenCV YOLOv5, an acronym for 'You Only Look Once', represents a significant advancement in the domain of computer vision models. Historically, object detection algorithms primarily focused on identifying larger objects without tailored optimization for smaller ones. YOLOv5 addresses this gap by effectively trimming the feature map output from the large object detection layer, thus enhancing computational efficiency and creating a more compact model. This model offers five variants - nano (n), small (s), medium (m), large (l), and extra-large (x), catering to a range of performance requirements from high speed and lower accuracy to slower processing with higher accuracy. Widely used for real-time object detection, YOLOv5's versatility makes it suitable for various applications, thanks to its balance of speed and efficiency.

OpenCV, which stands for Open-Source Computer Vision, is a comprehensive library designed for image and video processing tasks. Its design emphasizes computational efficiency, particularly in real-time contexts, enabling developers

to focus on application development without extensive optimization concerns. OpenCV's capabilities span across multiple core areas including image processing, video analysis, object detection and recognition, machine learning, camera calibration, and GUI support. Its compatibility with various interfaces and operating systems, alongside its multi-core processing capabilities, facilitates the creation of high-quality, commercial-grade software in the realms of computer vision and image processing.

React React, a widely recognized JavaScript library, is known for its declarative, efficient, and flexible approach to building user interfaces. It employs Webpack for the automatic compilation of React, JSX, and ES6 code, and efficiently manages CSS file prefixes. While React is more a library than a programming language, its extensive range of extensions for full application architecture support has made it a staple in web development. React's popularity stems from its ability to simplify the creation of dynamic applications through minimized coding, enhanced performance via Virtual DOM, component reusability, unidirectional data flow, and adaptability for both web and mobile app development. The combination of these features, along with a user-friendly development process, has led to its widespread adoption in both organizations and businesses.

Azure SQL Database & Docker Azure SQL, a suite of managed, secure, and intelligent products, leverages the SQL Server database engine within the Microsoft Azure cloud. As a relational database-as-a-service (DBaaS), Azure SQL Database builds on the familiar SQL Server engine, aligning with the Platform-as-a-service (PaaS) industry model. It offers a streamlined experience for application development, deployment, and scaling, thanks to its managed service approach. This includes automated tasks such as patching, backups, and monitoring. Key features like robust security through data encryption, high availability via automatic backups and geo-replication, compatibility with various applications, and comprehensive monitoring and management tools, position Azure SQL Database as a leading solution for modern cloud-based applications.

Docker, an open platform for application development, shipping, and running, revolutionizes the separation of applications from infrastructure through containerization. Containers encapsulate an application and its dependencies into a single, lightweight, portable unit. Docker's architecture, comprising Docker images (lightweight, standalone, executable packages), Docker containers (instances of Docker images), Docker Hub (a cloud-based registry service for sharing Docker images), and Dockerfile (a set of instructions for building an image), is fundamental in modern software development and deployment. Docker not only facilitates collaboration but also streamlines deployment workflows and boosts the scalability and reliability of applications.

2.3 Reasoning

Model Selection Rationale The selection of the YOLOv5x model as the linchpin for our object detection system was a deliberate choice, driven by its

exceptional processing speed and accuracy. YOLO outperforms other models in terms of both efficiency and velocity, nearly seven times faster than alternatives like Faster R-CNN, while maintaining a comparable level of precision. This choice aligns with our objective of achieving real-time object detection, where both speed and accuracy are essential. The YOLOv5x, the most extensive model in the YOLOv5 series, was particularly chosen for its enhanced capabilities in our application context.

Data Architecture and Flow The structure and flow of data are crucial for the operational efficacy of our system. NginX was strategically deployed for its superior data handling and rapid processing attributes. It skillfully manages client requests across server infrastructure, optimizing response times and ensuring the delivery of timely and accurate seating information to users. This strategic choice underscores our commitment to system efficiency and user-centric information provision.

System Architecture Design Our system's architectural design prioritizes real-time detection and analysis of seat occupancy. Cameras installed across various spaces continuously monitor seat availability, identifying occupied and vacant seats in real time. The collected data is then relayed to a central processing unit for analysis, following which, up-to-date seating information is promptly disseminated to users. This architecture enables the delivery of this vital information through various platforms, including mobile apps and websites, enhancing seat management efficiency and user guidance.

User Interface Design Employing React's responsive design, our user interface is crafted to deliver a smooth and consistent experience across diverse devices, from mobile phones to desktop computers. This approach ensures that users can access seating information with ease, streamlining the process of seat selection. The focus on a responsive and intuitive interface significantly contributes to enhancing user satisfaction and engagement with the application.

2.4 Challenges & Solution

Real-time Detection Real-time object detection poses a significant challenge, necessitating swift identification and processing of object positions and states in ever-changing environments, such as learning spaces. To address this, the 'Empty Seats?' application employs the YOLOv5 x model, renowned for its rapid inference capabilities. This enables near real-time processing of video feeds from cameras, providing an accurate reflection of the current occupancy status of seats.

View Transformation A common issue in fixed-position camera setups is the overlapping of objects in the field of view. To mitigate this, the application uses

a Bird's eye view transformation, applying a 2 x 3 transformation matrix. This technique adjusts the coordinates obtained from object detection, transforming them into an accurate top-down perspective. This process enhances the precision in locating and identifying objects within the monitored spaces.

Data Processing and Accuracy Ensuring accurate data processing is paramount for the system's reliability, particularly in determining seat occupancy. The challenge lies in accurately detecting whether a seat is occupied, as erroneous detection can lead to misinformation. The application tackles this by employing coordinate-based data processing; it utilizes the center points of detected objects, extract occupancy information of each seats and convert into database-compatible formats. This method significantly increases the accuracy in identifying occupied seats and effectively communicates this information to the users.

3 Implementation

3.1 Utilized Modules Implementation

RTSP Video Capture In this project, multiple IP cameras are utilized to capture real-time spatial data. These IP (Internet Protocol) cameras, commonly used in surveillance, stream video using the Real-Time Streaming Protocol (RTSP). RTSP facilitates efficient control and delivery of streaming media over IP networks. The streams from these cameras, accessed via URLs containing the camera's IP address, port, and stream identifier, provide the necessary video source for the YOLOv5 Object Detection model.

YOLOv5 Implementation The project leveraged the YOLOv5 x model, the most advanced variant in the YOLOv5 series, combined with the COCO dataset for object identification. The primary focus was on detecting predefined objects within the environment.

- **Target Objects for Detection**
  The objects targeted for detection included:
  - Individuals (class ID 0)
  - Book (class ID 73)
  - Laptop (class ID 63)
  - Television (class ID 62)
  - Remote (class ID 65)
  - Cell phone (class ID 67)
  - Mouse (class ID 64)
  - Keyboard (class ID 66)
  - Backpack (class ID 24)
  - Handbag (class ID 26)

– **Implementation Details**
  1. **Image Capture**: Images of the space were captured through a network of cameras, providing real-time visual data.
  2. **Object Detection**: Utilized the YOLOv5 x model for precise detection of the listed objects. The model's ability to accurately identify these objects played a critical role in the data collection process.
  3. **Data Processing**: Modified the `detect.py` script to return coordinates of detected objects as a list, enabling integration with the database for further processing.

– **Code Execution**
  The execution of the YOLOv5 model involved:
  1. Running a Python script to activate the object detection process.
  2. Utilizing pre-trained weights from the file `yolov5x.pt` for enhanced model accuracy.
  3. Setting the image resolution to 640 pixels, optimizing the balance between detection accuracy and computational efficiency.
  4. Applying a confidence threshold of 0.25 to ensure only high-probability detections are considered.
  5. Specifying the video source through RTSP, facilitating real-time detection from network camera feeds.
  6. Employing the `--classes` option to focus on the detection of specific object classes as mentioned above.

This comprehensive approach to implementing YOLOv5x model ensured accurate and efficient object detection, integral to the project's success.

Detected Coordinate Data Processing The project leverages Python OpenCV for implementing features such as comprehensive coordinate transformation using OpenCV. This implementation includes the application of a homogeneous matrix,made of numpy array, for transformations, along with necessary normalization and denormalization steps. The process yields Bird's eye view coordinates of detected objects (e.g. person, laptop, book) in the image or video

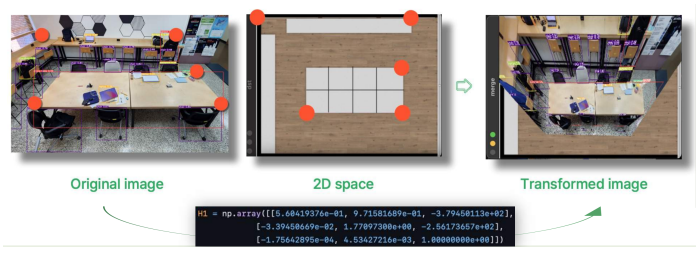

Fig. 2. Coordinate transformation process diagram.

– **Coordinate Transformation Logic**: Homogeneous coordinates are used for transformations, adding an extra dimension to (x, y) coordinates before applying the transformation. Post-transformation, coordinates revert to 2D by dividing by the third component.

$$\begin{bmatrix} t_i x'_i \\ t_i y'_i \\ t_i \end{bmatrix} = \texttt{map\_matrix} \cdot \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

$$dst(i) = (x'_i, y'_i), src(i) = (x_i, y_i), i = 0, 1, 2, 3$$

Fig. 3. Coordinate transform matrix formula

– **Parameters**
src Coordinates of quadrangle vertices in the source image.
dst Coordinates of the corresponding quadrangle vertices in the destination image.
– **Normalization and Denormalization**: To handle varying image sizes, normalized coordinates from the YOLOv5 model are adjusted accordingly, ensuring consistent transformations.
– **Visualization of Transformed Coordinates**: For debugging purposes, transformed coordinates can be visualized using OpenCV and Matplotlib, although this feature is not part of the overall data flow in the project.

As shown in Figure 2, the coordinate transformation process is integral to achieving accurate object positioning.

3.2 Frontend Implementation

The frontend development focused on creating a responsive web design using React and Material UI, ensuring seamless accessibility across a range of devices. The primary components of our web interface include:

Space List Page

– **Functionality**: This page displays a list of open spaces, providing users with real-time information about each space.
– **Data Fetching**: Utilizes the `/data/getSpace` API to retrieve data from the `SpaceTable` in the database.

- **Layout**: Information is presented in card format, showing the space's name, location, and current usage status.
- **Status Indicators**: Spaces are color-coded to indicate congestion levels - red for very crowded (over 80% occupancy), yellow for crowded (50% or more occupancy), and green for availability (less than 50% occupancy).
- **Navigation**: The 'Take a seat' button navigates users to a detailed seat layout page for the selected space.

Description Page

- **Purpose**: Provides a descriptive overview of each space, detailing the atmosphere and environment.
- **Navigation**: Like the Space List Page, clicking the "Take a Seat" button redirects users to a detailed seat layout page for the space.

Real-time Status Page

- **Functionality**: Allows users to view real-time seat layouts for each space.
- **Data Retrieval**: On accessing or refreshing the page, the server fetches data from the `SeatDataTable` using the `/data/getSeats` API.
- **Visualization**: Seat occupancy is color-coded, with red for occupied seats, yellow for reserved (items placed), and green for vacant seats.

These frontend components collectively enhance user experience by providing intuitive, real-time insights into space availability and features.

3.3 Backend Implementation

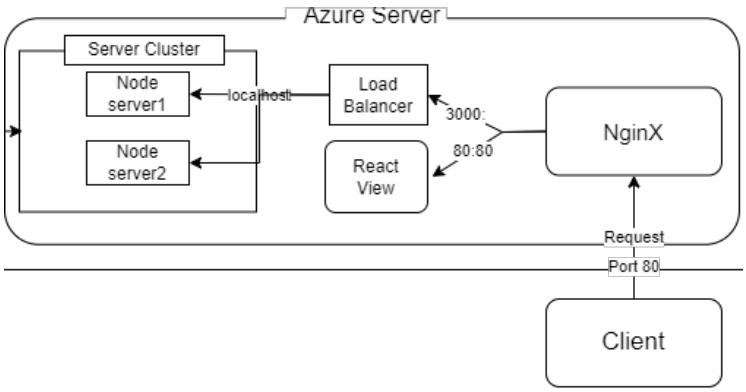

Fig. 4. Backend architecture diagram.

Node.js/Express.js Server The project employs Express.js, a minimalistic web framework for Node.js, to develop a robust API. Routes, such as `dataRoutes` imported from `./routes/data.js`, are managed modularly. The Express application is configured to parse JSON payloads, with the server listening on port 3000. Root route setup ensures functionality verification, and the entire server application is exportable for broad project use.

Data Route Management The project's data route management involves several key components, structured as follows:

- **Environment Variable Management**: Utilization of the `dotenv` package for securely managing environment variables. This ensures that sensitive information like database credentials is kept secure and separate from the codebase.
- **Database Interaction**: Use of the `mssql` module for handling interactions with the database. This module provides a reliable and efficient means of executing SQL queries within a Node.js environment.
- **Endpoint Definitions**:
  - `/data/getSpace`: An endpoint defined for retrieving data about different spaces. This route likely involves asynchronous operations, potentially including database queries to fetch detailed information about each space.
  - `/data/getSeats`: This endpoint is designed for querying seat data for each space. Similar to `/getSpace`, it indicates asynchronous operations that involve database interactions to determine the status of seats within a given space.

This structure ensures efficient and secure management of data routes, vital for the system's overall functionality and data integrity.

Nginx Nginx, integral to the project's architecture, is characterized by several key features:

- **High-Performance Web Server**: Recognized for its stability and resource efficiency, Nginx excels in managing high traffic and complex load scenarios.
- **Reverse Proxy Functionality**: It operates as a reverse proxy, redirecting traffic from port 80 to 3000. This setup is essential for routing user requests to the appropriate Node.js server or React.js frontend.
- **Static Content Management**: Nginx efficiently handles the delivery of static content, a vital aspect of web application performance.
- **Load Balancing**: Beyond routing, Nginx plays a pivotal role in load balancing. It evenly distributes server load, ensuring optimal performance and minimizing downtime.

These functionalities of Nginx contribute significantly to the robustness and efficiency of the web application's backend architecture.
Figure 4 illustrates the backend architecture, highlighting the roles of Node.js, Express.js and Nginx in the system.

3.4 Database Implementation

The project leverages an Azure SQL Database, a cloud-based relational database service, focusing on managing space and seating data. This database plays a crucial role in providing real-time information about various spaces and their seat occupancy status. Efficient SQL queries and database connection management in the Node.js backend are integral to harnessing the full potential of the Azure SQL Database, ensuring scalability and responsiveness.

Database Table The database consists of two primary tables:

Field	Type	Description
`name`	NVARCHAR(255)	Primary key, denotes the space's name.
`title`	NVARCHAR(255)	A unique title for each space.
`address`	NVARCHAR(255)	The physical address of the space.
`available_seat`	INT	The count of available seats in the space.
`total_seat`	INT	The total number of seats in the space.

Table 1. `SpaceTable` structure.

Field	Type	Description
`seatID`	INT	Primary key, uniquely identifies each seat.
`name`	NVARCHAR(255)	Links to `name` in `SpaceTable`.
`seatState`	INT	Indicates the current status of the seat (occupied, available, etc.).
`seatNumber`	INT	Identifies each seat in each space.

Table 2. `SeatDataTable` structure.

1. `SpaceTable`: Represents different spaces (like rooms or areas) information in the application.

2. `SeatDataTable`: Manages information about individual seats within spaces.

As depicted in Figure 5, the database structure is designed to efficiently handle space and seat data.

Implementation Strategy The database implementation involves several strategies:

– **Data Retrieval and Manipulation**: Utilizing the `mssql` module in Node.js for executing SQL queries, facilitating interaction with the database.

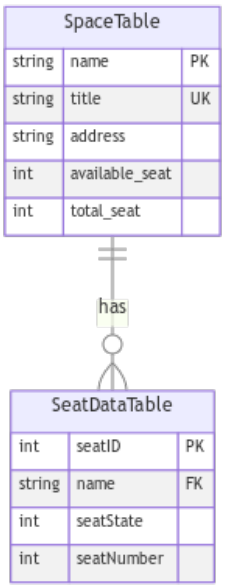

Fig. 5. ER diagram of database structure.

- **Example Use Cases**:
  - Fetching Available Seats: Retrieving `available_seat` from `SpaceTable` for specific spaces.
  - Seat Status Update: Modifying `seatState` in `SeatDataTable` based on occupancy changes.
  - Space-Seat Relationship Queries: Conducting join operations between `SpaceTable` and `SeatDataTable` for comprehensive data analysis.
- **API Endpoints**: Implementing endpoints like `/data/getSpace` and `/data/getSeats` to interact with the database, facilitating data retrieval for spaces and seats.

3.5 Deploy Implementation

Docker Configuration (`Dockerfile` and `docker-compose.yml`) The deployment strategy of the project is anchored in Docker, a powerful containerization platform. This section outlines the key aspects of Docker configuration:

- **Containerization Strategy**:
  The `Dockerfile` is designed to set up a Node.js environment, handle dependencies, and prepare the application for containerized deployment. This file is critical in defining the environment and commands needed for the Docker container.
- **Service Orchestration**:

- The `docker-compose.yml` file is used to define and run multi-container Docker applications. In this project, it specifies the orchestration of several services, including the Node.js server, React.js frontend, and an Nginx service. Each of these services is containerized with specific roles and configurations to work in unison.
- The deployment utilizes a structured directory named `WebHost` as the root of the orchestration. This directory contains subdirectories such as `app`, `nginx`, and `server`, each housing the respective build environments for these components.

This Docker-based approach ensures streamlined deployment and scalability of the web application, allowing each component to function efficiently in a coordinated manner.

4 Evaluation

4.1 Detection Execution Time and Data Processing Execution Time

Total Detection Execution Time

- **Graph Analysis**: Line graphs illustrate the execution time of object detection across different locations.
- **X-axis**: Represents each execution cycle, calculated per 10 frame.
- **Y-axis**: Displays the time taken for object detection in each cycle.
- **Locations**: The graphs include data for Parksangjo Lounge, Engineering Study Rooms, and Hae Dong Library.

Total Data Processing Execution Time

- **Graph Analysis**: Line graphs visualize the data processing execution time.
- **X-axis and Y-axis**: Similar to the detection time graph, representing cycles and execution time, respectively.
- **Locations**: Includes the same three locations for comparison.

Total Execution Time

- **Graph Analysis**: Line graphs visualize the total execution time
- **X-axis and Y-axis**: Similar to the detection time graph, representing cycles and execution time, respectively.
- **Locations**: Includes the same three locations for comparison.

As shown in Figure 6, the average execution time for total data processing is 8.03 ms, total detection execute time is 2.69 seconds, and total execution time is 2.70 seconds, all of which are within our initial target of under 3.0 seconds.

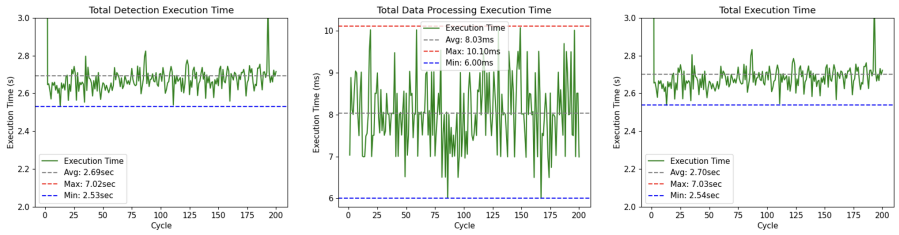

Fig. 6. Total detection, data processing, execution time

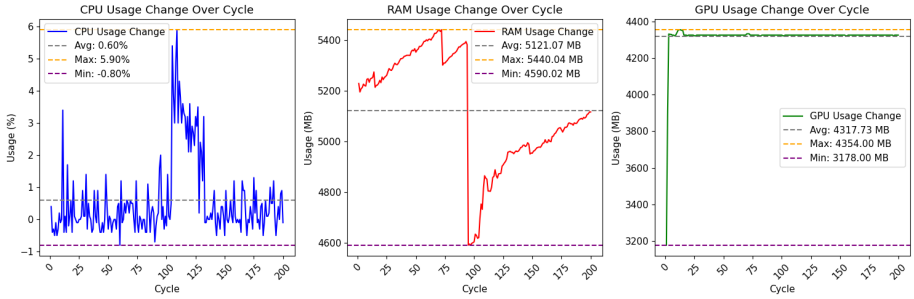

Fig. 7. CPU, RAM, GPU usage over cycle

4.2 Resource Usage Evaluation

GPU and RAM Usage

- **GPU Model**: Utilization of RTX 3070Ti for GPU-intensive tasks.
- **Performance Metrics**: Monitoring of time spent and RAM usage during processing.
- **Evaluation**: Details of resource usage efficiency and potential areas for optimization.
- **CPU Usage Change:**
  The CPU usage graph demonstrates fluctuating patterns across cycles, with peaks representing increased processing demand. The average CPU usage is relatively stable at 0.60%, with a maximum spike of 5.90% indicating occasional heavy processing tasks.
- **RAM Usage Change:**
  The RAM usage exhibits a steady increase over time, suggesting a possible memory leak or the accumulation of data in memory. The average RAM usage is 5121.07 MB, peaking at 5440.04 MB, which may necessitate investigation into memory management.
- **GPU Usage Change:**
  GPU usage is consistent, with a sharp incline at the start. This could be due to initial resource allocation followed by stabilization. The GPU maintains an average usage of 4317.73 MB, with a maximum of 4354.00 MB.

5 Limitation

While the project effectively implements real-time object detection using the YOLOv5 x model and the COCO dataset, it encounters several limitations:

1. **Dependency on Environmental Conditions**: The system's accuracy is significantly influenced by external factors such as lighting conditions and camera view obstructions. Inconsistent lighting or obstructed views can adversely affect the detection accuracy.
2. **Reliance on Predefined Desk Location List**: The system depends on a predefined list of desks location grids for identifying occupancy. This limitation could result in some items within a learning space being overlooked, leading to inaccurate occupancy detection.
3. **Camera Positioning and Angle Limitations**: The effectiveness of object detection and seat occupancy determination can be compromised by the cameras' positioning and angles. Although the view transformation algorithm attempts to mitigate this, it cannot fully rectify all positioning and angle-related limitations.
4. **Synchronization Challenges with Real-Time Data**: Maintaining synchronization between real-time data and the database is challenging, particularly given the need for a robust and reliable network infrastructure to ensure data integrity and timely updates.
5. **Scalability Constraints**: While designed for scalability, the system's actual scalability is bound by the limitations of physical hardware and network capacity.

These limitations underscore the necessity for continuous optimization and potential system upgrades to ensure enduring effectiveness and adaptability in various operational conditions and usage scenarios.

6 Related Work

6.1 SKKU Central/Samsung Library

SKKU Central and Samsung Library, the primary libraries on the Seoul and Suwon campuses, offer an advanced reservation system. This system, accessible via both a website and mobile applications (compatible with Android and iOS), enables students to check and book available seats remotely. The libraries employ a student ID card system for entry and exit, allowing real-time tracking of presence and the automatic freeing up of seats based on absent time.

6.2 HAE Dong Academic Information Room

The HAE Dong Academic Information Room, available to students of the College of Information and Communication Engineering (CICE), is segmented into various spaces including lounges, seminar rooms, and study areas. Despite the availability of a reservation system for seminar rooms, no such system exists for the other spaces, posing challenges in seat availability verification before visiting.

6.3 SKKU Campus Map

The SKKU Campus Map provides comprehensive information about the campus, including buildings, facilities, and study spaces. While it offers valuable insights and tips, including the total number of seats in study areas, it lacks real-time seat availability data. This limitation can lead to student inconvenience, as they may need to physically verify space availability.

7 Conclusion

'Empty Seats?' represents a significant advancement in real-time seat occupancy monitoring, allowing users to efficiently check the availability of seats in open spaces. Successfully tested in various locations within Sungkyunkwan University's Natural Science Campus, including Parksangjo Lounge, Haedong Library, and Engineering Building study rooms, 'Empty Seats?' demonstrates rapid and accurate seat occupancy assessments.

The system's scalability is notable. With the installation of cameras and the setup of necessary parameters like the perspective transform matrix H values and seat division grid, 'Empty Seats?' can be readily expanded to additional locations. This project not only enhances space utilization but also brings efficiency to both users and administrators, streamlining the management and use of campus spaces. Its implementation stands as a testament to the potential of integrating technology into campus management, setting a new standard for space utilization efficiency.

References

1. OpenCV Documentation. *Geometric Image Transformations*, OpenCV 3.4 Documentation, `https://docs.opencv.org/3.4/da/d54/group__imgproc__transform.html#ga8c1ae0e3589a9d77fffc962c49b22043`. Accessed on Dec, 10, 2023.
2. YOLO. Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 779-788).
3. SKKU CAMPUSMAP. Available at: `https://eng.skku.edu/eng/About/campusinfo/CampusMap.do`.
4. YOLOv5 GitHub Repository. Available at: `https://github.com/ultralytics/yolov5.git`.
5. HAE Dong Studyroom Reservation System. Available at: `https://scg.skku.ac.kr/seminar/`.